**INT201 Client-side**

**Document Object Modeling (DOM)**

**Document Object Model (DOM)**

* Understanding the Document Object Model in JavaScript
* Hierarchy Of Nodes
* Node Types and Relationships
* Traversing Nodes
* Manipulating Nodes
* 4 Key Node Types (Document, Element, Text, Attribute)

**Document Object Modeling (DOM)**

* One of the most important objects in client side JavaScript programming is the Document object which comprise the structure and content of a document on the web.
* The Document Object Model (DOM) represents the page so that programs can change the document structure, style, and content.
* A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document, but the Document Object Model (DOM) representation allows it to be manipulated.
* As an object oriented representation of the web page, it can be modified with a scripting language such as JavaScript.
* The DOM is an application programming interface (API) for HTML and XML documents.
* The DOM represents a document as a **hierarchical tree of nodes**, allowing developers to add, remove, and modify individual parts of the page.
* The DOM is now a truly cross-platform, language-independent way of representing and manipulating pages for markup.
* The DOM is not a programming language, but without it, the JavaScript language wouldn't have any model or notion of web pages, HTML documents, XML documents, and their component parts.

Text

Description automatically generated**<script> Tag Placement**

* Including all JavaScript files in the <head> of a document means that all of the JavaScript code must be downloaded, parsed, and interpreted before the page begins rendering
* Graphical user interface, text

  Description automatically generatedFor pages that **require a lot of JavaScript code**, this can cause a noticeable delay in page rendering, during which time the browser will be completely blank.
* For this reason, modern web applications typically include all JavaScript references in the <body> element, after the page content.

**Document Object Modeling (DOM)**

* When an HTML document is loaded into a web browser, it becomes a **document object,** everything is a **node**
* The document object is the root node of the HTML document and all other nodes: element nodes, text nodes, attribute nodes, and comment nodes
* Each node type has different characteristics, data, and methods, and each may have relationships with other nodes.
* These relationships create a hierarchy that allows markup to be represented as a tree, rooted at a particular node.

Diagram

Description automatically generated

Table

Description automatically generated**Node types are represented by one of the following 12 numeric constants on the Node types:**

* The **Node** interface is implemented in JavaScript as the Node type, which is accessible in all browsers except Internet Explorer.
* All node types inherit from Node in JavaScript, so **all node types share the same basic properties and methods.**
* Every node has a **node Type property** that indicates the type of node that it is.

Text

Description automatically generated with low confidence

A picture containing diagram

Description automatically generated

1. **Node Relationships**

* All nodes in a document have relationships to other nodes.
* These relationships are described in terms of **traditional family relationships** as if the document tree were a family tree.
* In HTML, the <body> element is considered a child of the <html> element; likewise the <html> element is considered the parent of the <body> element.
* The <head> element is considered a sibling of the <body> element, because they both share the same immediate parent, the <html> element.

The value of someNode.firstChildis always equal to someNode.childNodes[0], and the value of someNode.lastChildis always equal to someNode.childNodes [someNode.childNodes.length - 1].

Diagram

Description automatically generated

With all of these relationships, the **childNodes property** is really more of a convenience than a necessity because **it’s possible to reach any node in a document tree** by simply using the relationship pointers.

**Traversing Nodes**

**Element Traversal**

1. The Element Traversal API adds five new properties to DOM elements:

* childElementCount—Returns the number of child elements (excludes text nodes and comments).
* firstElementChild—Points to the first child that is an element. Element-only version of firstChild.
* lastElementChild—Points to the last child that is an element. Element-only version of lastChild.
* previousElementSibling—Points to the previous sibling that is an element. Element-only version of previousSibling.
* nextElementSibling—Points to the next sibling that is an element. Element-only version of nextSibling.

Text, letter

Description automatically generated

Text, letter

Description automatically generated Text

Description automatically generated

\*The firstChild and lastChild return the first and last child of a node, which can be any node type including text node, comment node, and element node.

\*\*The firstElementChild and lastElementChild return the first and last child Element node.

**Selecting Elements**

The Document type provides two methods to specific element or sets of elements to perform certain operations.

* getElementById() //returns the element if found, or null if an element with that ID doesn’t exist.
* getElementsByTagName() //returns an HTMLCollection of elements with the given tag name.
* getElementsByName() //returns a NodeList, which returns all elements that have a given name attribute.
* getElementsByClassName() //returns a NodeList containing all elements that have all of the specified classes applied.
* querySelector() //a CSS query and returns the first descendant element that matches the pattern or null if there is no matching element.
* querySelectorAll() //the CSS query and returns all matching nodes instead of just one. This method returns a static instance of NodeList.

**An HTMLCollection** is a collection of HTML elements.

**A NodeList** is a collection of document nodes.

Both an HTMLCollection object and a NodeList object is an array-like (collection) of objects.

Graphical user interface, text

Description automatically generated

**Manipulating Nodes**

* appendChild() //adds a newly node to the end of the childNodes list
* createElement() //create a new HTML element
* insertBefore(newNode, referenceNode) //The node to insert becomes the previous sibling of the reference node
* replaceChild(newChild, oldChild) //replaces a child node within the given (parent) node
* removeChild(child) //removes a child node from the DOM and returns the removed node.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Nodes Types**

**Node.nodeType**

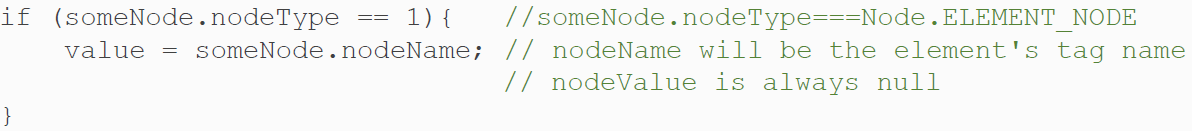
Node.nodeType property is an integer that identifies what the node is. It distinguishes different kind of nodes from each other, such as elements, text and comments.

Graphical user interface

Description automatically generated

**The nodeName and nodeValue Properties**

* Two properties, nodeName and nodeValue, give specific information about the node.
* The values of these properties are completely dependent on the node type.
* It s always best to test the node type before using one of these values, as the following code shows:



**Document Type**

**The Document Type**

* JavaScript represents document nodes via the Document type.
* The Document type can represent HTML pages or other XML-based documents.
* The document object, as an instance of HTMLDocument, has several additional properties that standard Document objects do not have.
* The document object can be used to get information about the page and to manipulate both its appearance and the underlying structure.

A **Document node** has the following characteristics:

* nodeType is 9.
* nodeName is "#document”.
* nodeValue is null.
* parentNode is null.
* child nodes may be Element (maximum of one), ProcessingInstruction, or Comment.
* appendChild(), removeChild(), and replaceChild() methods aren’t used on document because the document type (if present) is read only and there can be only one element child node.

1. **Document Children**

* This example shows that the values of documentElement, firstChild, and childNodes[0] are all the same—all three point to the <html> element.
* As an instance of HTMLDocument, the document object also has a body property that points to the <body> element directly.

A picture containing text

Description automatically generated

1. **Document Information**

* The first such property is title, which contains the text in the <title> element and is displayed in the title bar or tab of the browser window.

Text

Description automatically generated

1. **Special Collections**

* document.anchors—contains all <a>elements with a name attribute in the document.
* document.forms—contains all <form> elements in the document. The same as document.getElementsByTagName("form").
* document.images—contains all <img> elements in the document. The same as document.getElementsByTagName("img").
* document.links—contains all <a>elements with an href attribute in the document.

**Element Type**

* The **Element type** represents an XML or HTML element, providing access to information such as its tag name, children, and attributes.
* An Element node has the following characteristics:
  + nodeTypeis 1.
  + nodeNameis the element’s tag name.
  + nodeValueis null.
  + parentNodemay be a Document or Element.
  + Child nodes may be Element, Text, Comment, ProcessingInstruction, CDATASection, or EntityReference.
* An element’s tag name is accessed via the nodeName property or by using the tagName property; both properties return the same value.

Graphical user interface, text, application

Description automatically generated

**HTML Elements**

* All HTML elements are represented by the HTMLElement type, either directly or through subtyping.
* The HTMLElement inherits directly from Element and adds several properties.
* Each property represents one of the following standard attributes that are available on every HTML element:
* id—A unique identifier for the element in the document.
* title—Additional information about the element, typically represented as a tooltip.
* lang—The language code for the contents of the element (rarely used).
* dir—The direction of the language, "ltr" (left-to-right) or "rtl" (right-to-left); also rarely used.
* className—The equivalent of the class attribute, which is used to specify CSS classes on an element. The property could not be named class because class is an ECMAScript reserved word.

Graphical user interface, text, application

Description automatically generated

The three primary DOM methods for working with attributes are getAttribute(), setAttribute(), and removeAttribute().

Text

Description automatically generated

**Creating Elements**

* New elements can be created by using the document.createElement() method.
* This method accepts a single argument, which is the tag name of the element to create.
* In HTML documents, the tag name is case-insensitive
* To create a <div> element, the following code can be used:



* The element can be added to the document tree using appendChild(), insertBefore(), or replaceChild().
* The following code adds the newly created element to the document s <body> element:



* Once the element has been added to the document tree, the browser renders it immediately.

Any changes to the element after this point are immediately reflected by the browser.

**Text Type**

* Text nodes are represented by the Text type and contain plain text that is interpreted literally and may contain escaped HTML characters but no HTML code.
* A Text node has the following characteristics:
  + nodeType is 3.
  + nodeName is "#text".
  + nodeValue is text contained in the node.
  + parentNode is an Element.
* child nodes are not supported.
* The text contained in a Text node may be accessed via either the nodeValue property or the data property, both of which contain the same value.
* Changes to either nodeValue or data are reflected in the other as well.

The following methods allow for manipulation of the text in the node:

* appendData(text) - Appends text to the end of the node.
* deleteData(offset, count) - Deletes count number of characters starting at position offset.
* insertData(offset, text) - Inserts text at position offset.
* replaceData(offset, count, text) - Replaces the text starting at offset through offset + count with text.
* splitText(offset) - Splits the text node into two text nodes separated at position offset.
* substringData(offset, count) - Extracts a string from the text beginning at position offset and continuing until offset + count.
* length property or nodeValue.length or data.length returns the number of characters in the node.

Text

Description automatically generated with medium confidence

The following code lets you access this node:



Once a reference to the text node is retrieved, it can be changed like this:



**Creating Text Nodes**

* New text nodes can be created using the document.createTextNode() method, which accepts a single argument the text to be inserted into the node.
* As with setting the value of an existing text node, the text will be HTML or XML encoded, as shown in this example:



* it does not appear in the browser window until it is added to a node in the document tree. The following code creates a new <div> element and adds a message to it:

Text

Description automatically generated

**Attr Type**

* Element attributes are represented by the Attr type in the DOM.
* The Attr type constructor and prototype are accessible in all browsers.
* Technically, attributes are nodes that exist in an element’s attributes property.
* Attribute nodes have the following characteristics:
  + nodeType is 2.
  + nodeName is the attribute name.
  + nodeValue is the attribute value.
  + parentNode is null.
  + Child nodes are not supported in HTML.
  + Child nodes may be Text or EntityReference in XML.
* Even though they are nodes, attributes are not considered part of the DOM document tree.
* Attribute nodes are rarely referenced directly, with most developers favoring the use of getAttribute(), setAttribute(), and removeAttribute().

**Creating Attribute Nodes**

New attribute nodes can be created by using document.createAttribute() and passing in the name of the attribute. For example, to add an align attribute to an element, the following code can be used:

Text

Description automatically generated

**Note**

* The Attr node inherits from Node but is not considered a part of the document tree.
* Common Node attributes like parentNode, previousSibling, and nextSibling are null for an Attr node. You can, however, get the element to which the attribute belongs with the ownerElement property.
* Note that this is different from the W3C XML Document Object Model (DOM), which does not treat the element with an attribute as the parent of the attribute.

**System Dialogs** //07\_SystemDialogs

* The browser is capable of invoking system dialogs to display to the user through the **alert(), confirm(),** and **prompt()** methods.
* These dialogs are not related to the web page being displayed in the browser and do not contain HTML.
* Their appearance is determined by operating system and/or browser settings rather than CSS.
* A picture containing shape

  Description automatically generatedAdditionally, each of these dialogs is synchronous and modal, meaning code execution stops when a dialog is displayed, and resumes after it has been dismissed.

**System Dialogs: alert()**

* The alert() method simply accepts a string to display to the user.
* When alert() is called, a system message box displays the specified text to the user, followed by a single OK button.
* A picture containing graphical user interface

  Description automatically generatedAlert dialogs are typically used when users must be made aware of something that they have no control over, such as an error.

**System Dialogs: confirm()**

* A confirm() method looks similar to an alert dialog in that it displays a message to the user.
* The main difference between the two is the **presence of a Cancel button along with the OK button**, which allows the user to indicate if a given action should be taken.
* The confirm() method returns true if the user clicked "OK", and false otherwise.

**System Dialogs: prompt()**

* The final type of dialog is displayed by calling prompt(), which along **with OK and Cancel buttons**, this dialog has a text box where the user may enter some data.
* Graphical user interface, application

  Description automatically generatedThe prompt() method accepts two arguments: the *text to display* to the user, and the *default value* for the text box (which can be an empty string).
* If the OK button is clicked, prompt() returns the value in the text box; if Cancel is clicked or the dialog is otherwise closed without clicking OK, the function returns null.